SASM Cross Assembler

User's Guide

Version 1.35

Table of Contents

CHAPTER 1: Introduction 4
Main Features 4
Invoking SASM 4
Options summary 4
Source File 5
Output Files 5

CHAPTER 2: Program Structure 6
Assembler Source Line Format 7
Label 7
Mnemonic 7
Operand 7
Comment 7
Constants 7
Characters or String Constants 7
Numeric Constants 8
Symbols 8
Symbol Names 8
Reserved Symbols 9
Expressions 10
Arithmetic Operators 10
Well-defined Expressions 10

CHAPTER 3: SASM Assembler Directives 11
DEVICE - Define device type and fuse bits ..... **Error! Bookmark not defined.**
ID - Define ID string up to 8 characters 12
RESET - Set reset vector address 17
EQU - Equate a symbol to an expression ..... **Error! Bookmark not defined.**
SET - Set a symbol equal to an expression 17
DS - Define space in program memory 17
DW - Define data in memory 17
RES - Reserve storage in memory 18
INCLUDE - Insert external source file 18
ORG - Set program origin 18
IF.ELSE.ENDIF - Conditional assembly 18
IFDEF.ELSE.ENDIF - Conditional assembly 18
IFNDEF.ELSE.ENDIF - Conditional assembly 18
REPT ENDR - Repeat code block 20
LPAGE - Insert page eject in listing file 20
SPAC - Insert lines in listing file 20
TITLE - Define program heading 21
END - End of source program 21

CHAPTER 4: Using Macros 22
Macro Definition 22
Macro Heading 22
Macro Body 22
Macro Terminator 23
Macro Call 23
Parameters 23
Local Symbols 23
Macro Example 24

Definition 24
Macro Call 24
Macro expansion 24

CHAPTER 5: SASM Assembler Output Files 25
Object File (HEX or OBJ) 25
Listing File (LST) 25
Example 25
Cross Reference Listing 26
Symbol File (SYM) 26
Map File (MAP) 26
Error File (ERR) 26
Error Messages 26

APPENDIX A: Summary of SX-based Instruction Set 27
Arithmetic and Shift Operations 27
Bitwise Operations 27
Data Movement Operations 27
Control Transfer Operations 28
System Control Operations 28

APPENDIX B: Object File Format 29
Intel Hex file formats 29
INHX8M: Merged 8-bit Intellex Hex Format 29
INHX16: 16-bit Hex Format 30
INHX8S: Split 8-bit Intellec Hex file format 30
Binary file Format 30

APPENDIX C: SXREG.INC Definition File 31

APPENDIX D: Error Message 32

CHAPTER 1: Introduction

This User's Guide describes the SASM Assembler for the SX-based microcontrollers from Scenix Semiconductor Inc.

The guide explains how to invoke and use SASM. Topics include program structure, directives, macros and file outputs. A summary on the SX basic instruction set is also given.

SASM Assembler is a software development tool that accepts the SX symbolic assembly language as input and translates it into object codes under the MS-DOS operating system on the IBM PC or compatible systems.

Main Features

- Translates programs (source code) written in SX Assembly language to machine executable code (object code) on IBM PC or compatibles running MS-DOS version 3.0 or higher
- Generates object code for Scenix's SX microcontroller families including the SX18AC, SX20AC, SX28AC, SX48BD, and SX52BD in 4 different formats: INHX8M, INHX16, INHX8S and BIN16
- Provides MACRO and conditional assembly capabilities
- Supports Hex, Decimal (default) and Binary source and listing formats

Invoking SASM

Use an editor of your choice to create an ASM source file. Assemble this source file by typing the following at the command prompt of the directory where SASM.EXE resides:

```
SASM [options] file[.asm]        [Enter]
```

where file = source file name

Options summary

Opt	Arguments	Description	Default
/F	[INHX8M|INHX16|INHX8S|BIN16]	Output Format	INHX8M
/P	[SX18AC|SX20AC|SX28AC|SX48BD |SX52BD]	Processor Type	SX18AC
/R	[HEX|DEC|OCT]	Default Radix	DEC
/T	[TABWIDTH]	Tab width	8
/W	[0|1|2]	Warning level	1
/I	Turn on Case sensitivity	Symbols	Off
/L		No program listing	Off
/H or /?		Display Help Message	

NOTE 1: To eliminate comments (e.g. crossing page boundary) from the list files, set warning to a higher level. For example, set /W to 2. /W 0 will include all comments, warning errors and severe errors.

/W 1 will include warning errors and severe errors.
/W 2 will include severe errors only.

NOTE 2: It is recommended to set the processor type inside the main program rather than have it defined on-line during compilation. That is, include the following line in the .ASM file:

```
DEVICE     SX18AC

   Or

DEVICE     PINS18
```

Source File

Source file is the file to be assembled. SASM assumes all source files have an .ASM extension. If not, the entire filename, including extension, has to be provided at the command line.

Output Files

SASM Assembler outputs different files with the following extensions:

```
HEX      - Intel 8-bit Merged Hex file (*Default file format)
OBJ      - Binary object file
HXH/HXL  - Address/Data pairs for high-order and low-order 8 bits
           (only when INHX8S format is selected as output)
LST      - Program listing file
SYM      - Symbol file used for defining watch variables and setting
           breakpoints at label address. Used for symbolic or source-level
           debugging.
MAP      - Map file used for source-level debugging
ERR      - Error message file
```

CHAPTER 2: Program Structure

The structure of a source program consists of one or more statements and comments. Each statement can be a combination of mnemonics, directives, macros, symbols, expressions and/or constants.

Example assembly program

```
;       FILE:   DEMO28.ASM
;       DATE:
;       05-03-1999

        DEVICE  PINS28, PAGES4, BANKS8,
        DEVICE  OSCHS, TURBO, WATCHDOG
        ID      'Demo28'
        RESET   INI

; Define Symbols
FLAG    EQU     8
RDLY0   EQU     15h
RDLY0   EQU     16h
XPCH    EQU     0Bh
BUFH    EQU     18h
BUFL    EQU     19h

;=====================================================================
; Program Begin
;=====================================================================

                ORG     1

; Subroutine
WT100MS         CLR     RDLY0           ; clear RDLY0
WT100MX         CLR     !WDT            ; clear watchdog timer
                DECSZ   RDLY1
                JMP     WT100MX
                DECSZ   RDLY0
                JMP     WT100MX
                RETW    0

; Subroutine
WT16MS          MOV     W,#0C7H
                JMP     SETOPT
WT1MS           MOV     W,#0C3H
SETOPT          MOV     !OPTION,W
                CLR     RTCC
WT1MSX          CLR     !WDT
                MOV     W,#.250
                MOV     W,RTCC-W
                SC
                JMP     WT1MSX
                RETW    0

; Subroutine
KEYTBL          AND     W,#07H
                ADD     PC,W
                RETW    09BH            ; key code 0
                RETW    09BH            ; key code 1
                RETW    08AH            ; key code 2
                RETW    082H            ; key code 3
                RETW    074H            ; key code 4
                RETW    067H            ; key code 5
                RETW    062H            ; key code 6
                RETW    057H            ; key code 7
```

Assembler Source Line Format

The general format for a program source line is as followed:

[<Label>] <Mnemonic> [<Operand>] [<Comment>]

Label

The label field must begin at column one of the source line and is terminated by the first white space (a space, tab, or end-of-line character).

A label is optional and consists of 1 to 32 alphanumeric characters. It must begin with a letter, '_', '@' or ':' and may contain any combination of letters, digits, or underscores. Labels with more than 32 characters will be truncated.

Labels are generally used as a symbolic reference to program memory locations in the source code. A label may be the only field in the statement.

Mnemonic

The mnemonic field begins after the first white space in the source line and is terminated by the next white space. The field may contain an instruction mnemonic, assembler directive or macro.

Operand

The operand field begins immediately after the first white space following the mnemonic field and ends at the next white space. The field may contain one or more constants or expressions separated by commas.

Comment

The comment begins immediately after the first white space following the operand field, or the mnemonic field for those Mnemonics that do not require any operands. This is an optional field containing printable characters. Anything to the right of a semicolon (;) is treated as a comment and will be ignored by the assembler.

Constants

Constants are strings or numbers that SASM interprets as a fixed numeric value. SASM supports radix forms character, hexadecimal, decimal, and binary. SASM uses decimal as the default radix which helps determine what value will be assigned to constants in the object file when they are not explicitly specified by a base descriptor.

Characters or String Constants

String constants always begin with a single or double quote, and end with a matching single or double quote. SASM converts the characters between the quotes to ASCII values. For example,

```
MOV     W,#'A'
RETW    #'A'
```

Numeric Constants

A numeric constant in SASM consists of an arbitrary number of alphanumeric characters. The actual value of the constant depends on the radix you select to interpret it. Radices available in SASM are binary, octal, decimal, and hexadecimal, as shown in below. If no radix is given, SASM uses the default decimal radix.

Character	Radix
B	Binary
D	Decimal (default)
H	Hexadecimal

Hexadecimal numbers must always start with a decimal digit (0-9). If necessary, put a leading 0 at the left of the number to distinguish it between hexadecimal numbers that start with a letter (A- F). The hexadecimal digits A through F can be either upper- or lower-case. Constants can be optionally preceded by a plus or minus sign.

The formats for declaring a constant are detailed below. The base descriptor is case insensitive.

Type	Syntax	Example
Binary	<binary digit>B %<binary digit>	11111011B %11111011
Decimal	<digit> <digit>D	251(default radix) 251D
Hexadecimal	<hex digit>H 0x<hex digit> $<hex digit>	0FBH 0xFB $FB
Character	'<character>'	'A'

Symbols

A symbol represents a value, which can be a variable, an address label or an operand to an assembly instruction or directive.

Symbol Names

Symbol names are user-defined or predefined combination of letters (both uppercase and lowercase), digits and special characters. They are represented by a string of 1-32 alphanumeric characters with the first character being 'A'-'Z', 'a'-'z', '_', '@' or ':'. Valid characters for SASM are as follows:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
1 2 3 4 5 6 7 8 9 @ ! _
```

NOTE: SASM accepts upper and lower case characters, and defaults as case INSENSITIVE. Use /I command-line switch to turn on case sensitivity.

Symbol Types

Each symbol has a type that describes the characteristics and information associated with it. The way you define a symbol determines its type. SASM supports three symbol types:

DATA	a user-defined symbol that represents a data variable defined by EQU directive
VAR	a user-defined symbol that represents a data variable defined by SET directive
ADDR	a user-defined symbol that represents a code address or program counter location
RESV	a predefined symbol used internally by SASM

User-defined Symbols

Symbols are used in both label and operand fields in the source statement. Symbols are defined in the label field as either the current program address or as the resulting value of an EQU or SET expression. These values can then be used symbolically in operand fields. All symbols must be defined at some point in the source code by appearing in the label field. Please refer also to Appendix B for the SXREG.H definition file.

Reserved Symbols

The assembler has internally defined the following reserved symbols

=	DS	RES	DW
EQU	ORG	END	SET
ENDM	EXITM	IF	IFDEF
IFNDEF	ELSE	ENDIF	EXPAND
NOEXPAND	LIST	DEVICE	ID
RESET	SPAC	ZERO	LOCAL
LPAGE	MACRO	RADIX	TITLE
STITLE	INCLUDE	SUBTITL	LIST
PROCESSOR			

W	M	OR	PC
RA	RB	RC	RD
RE	RL	RR	SB
SC	SZ	ADD	AND
CLC	CLR	CLZ	DEC
INC	JMP	MOV	NOP
NOT	RET	SNB	SNC
SNZ	SUB	WDT	XOR
BANK	CALL	CLRB	DATA
MODE	PAGE	RETI	RETP
RETW	SETB	SKIP	SWAP
TEST	DECSZ	INCSZ	IREAD
MOVSZ	RETIW	SLEEP	OPTION

Expressions

Expressions are used in the operand field of the source statement and may contain constants, symbols or any combination of constants and symbols separated by operators.

Arithmetic Operators

The arithmetic operators available in expressions are as follows:

OPERATOR	DESCRIPTION	EXAMPLE
$	Current Program Counter	
+	Addition	1 + 2
+	Unary Plus	+ x
-	Subtraction	1 - 2
-	Unary Minus	- x
*	Multiplication	3*4
/	Division	3/4
<<	Left Shift	3 << 4
>>	Right Shift	3 >> 4
( )	Parentheses	((3 + 4) /5)
==	Logical Equal	x == y
!=	Logical Not Equal	x != y
<	Less than	3 < 5
>	Greater than	3 > 5
<=	Less than or equal	3 <= 5
>=	Greater than or equal	3 >= 5
!	Not	!(x== y)
~	Complement	~ x
|	Inclusive OR	x|y
&	Inclusive AND	x&y
||	Logical OR	x||y
&&	Logical AND	x&&y
^	Exclusive OR	x^ y

NOTE:

a. Associativity is left to right.

b. Nesting of parentheses may be used up to any level.

c. All operations use integer values, therefore, all fractions resulting from divisions will be truncated.

Well-defined Expressions

Some of the directives require well-defined expressions. These are expressions that can be evaluated on the first pass. This means any symbols used in the expression must be previously defined. For Example:

```
REG8        EQU   8h
            INC   REG8+1      ; fr = 9h
            INC   REG*2       ; fr = 10h
```

Expressions are used in the operand field of the source line and may contain constants, symbols, or any combination of constants and symbols separated by arithmetic operators.

Each constant or symbol may be preceded by one of the following:

'+' represents a positive value (default)

'-' represents a unary minus operation (2's compliment)

CHAPTER 3: SASM Assembler Directives

Directives are assembler commands that appear in the source code but are not translated directly into opcodes. They are used to control the program counter, allocation, and format listing outputs. Listed below is a brief summary of all the directives.

Directive	Description	Syntax
DEVICE	Define device type and fuse options.	DEVICE setting {setting,…}
ID	Define an ID string up to 8 characters.	ID 'string'
RESET	Define reset vector (starting location) of program.	RESET label
EQU	Equate a symbol to an expression. The symbol cannot be reassigned.	Symbol EQU expression
SET or =	Set a symbol equal to an expression. The symbol can be reassigned to new value.	Symbol SET expression Symbol = expression
DS	Define memory space by incrmenting the program memory address.	Symbol ds 1 Symbols ds 3
DW	Define 16-bit data in program memory.	DW data,{data…}
RES	Reserve storage in memory.	RES expression
INCLUDE	Insert external source file.	INCLUDE 'file'
ORG	Set program origin.	Set program origin
IF {ELSE} ENDIF	Conditional assembly.	IF expression {ELSE} ENDIF
IFDEF {ELSE} ENDIF	Conditional assembly.	IFDEF symbol {ELSE} ENDIF
IFNDEF {ELSE} ENDIF	Conditional assembly.	IFNDEF symbol {ELSE} ENDIF
REPT ENDR	Repeat block of program code a specified number of times.	REPT count ENDR
MACRO {EXITM} ENDM	Defines a macro.	Label MACRO {argument,…} {EXITM} ENDM
EXPAND or NOEXPAND	Specifies whether to expand the macro instructions in the list file.	EXPAND or NOEXPAND
LPAGE	Insert page eject in listing file.	LPAGE
SPAC	Insert lines in listing file.	SPAC expression
TITLE	Define program heading.	TITLE 'file'
END	Mark the End of source code.	END

Note: If you have declared the same directive more than once, the latter one will overwrite the previous definition.

DEVICE - Define device type and fuse bits

```
Syntax:          Device      settings {,settings…}
Description:     Specifies the device type and fuse bits of both FUSE and FUSEX
                 words to SASM assmebler.
Example:         DEVICE      PINS28,BANKS8,OSCHS
                 DEVICE      TURBO,STACKX,OPTIONX,CARRYX,PROTECT
```

There are different fuse settings for different device types.

Note: When debugging any SX18/20/28AC applications with the SX-ISD Debugger, the program memory size fuse bit must be set to BANKS8 (with 2K program memory).

The SX18/20/28AC device part is defaulted as Rev 5.2 or later version.
For backward compatibility, define OLDREV to indicate the SX18/20/28AC device is an Old Revision part. (Earlier than Rev 5.2):

```
DEVICE      PINS28, OLDDEV, BANKS8, PAGES8, OSCHS
```

For SX18/20/28AC Rev 5.2 or later, the fuse table is as follows:

Fuse Settings for SX18/20/28AC Rev 5.2 or later	Descriptions	Function	Default
PINS18/SX18AC PINS20/SX20AC PINS28/SX28AC PINS48/SX48BD PINS52/SX52BD	SX18AC SX20AC SX28AC SX48BD SX52BD	Specifies device type.	PINS18
BANKS1 BANKS2 BANKS4 BANKS8	1 page, 1 bank 1 page, 2 banks 4 pages,2 banks 4 pages,8 banks	Configure memory size. Should not be changed unless to reduce the amount of program memory in device.	BANKS8
OSCLP1 OSCLP2 OSCXT1 OSCXT2 OSCHS OSCRC	Ext Osc - LP1 Ext Osc - LP2 Ext Osc - XT1 Ext Osc - XT2 Ext Osc - HS Ext OSC - RC	Specifies external crystal / resonator Or external RC circuit.	OSCRC
IRCDIV1 IRCDIV4 IRCDIV3125 IRCDIV125	Int RC Osc - 4MHz Int RC Osc - 1MHz Int RC Osc - 128kHz Int RC Osc - 32kHz	Specifies interal oscillator divider	4MHz
IFBD	0 an ext feedback resistor is required between OSC1 and OSC2 pins. 1 crystal/resonator osc can rely on int feedback resistor between OSC1 and OSC2 pins	Internal Feedback Disable	Enable internal feedback resistor
BOR42 BOR26 BOR22 BOROFF	Brown-out reset at 4.2V Brown-out reset at 2.6V Brown-out reset at 2.2V Disable Brown-out reset	Specifies brown-out reset function and threshold voltage	Disable brownout
TURBO	0 Turbo mode(1:1) 1 Compatible mode(1:4)	Specifies turbo mode	Compatible mode
OPTIONX	0 8-bit option register and 8-level stack 1 6-bit option register and 2-level stack	Specifies Option register and stack extension	6 bits and 2-level
CARRYX	1 ignore carry flag as input to ADD and SUB instruction	ADD and SUB instructions use Carry flag as input	Carry flag ignored
SYNC	0 Enable synchronous inputs 1 Disable synchronous inputs	Enable or Disable synchronous input mode(for turbo mode operation).	Disabled
WATCHDOG	0 Disable watchdog timer 1 Enable watchdog timer	Enable or Disable Watchdog Timer.	Disabled
PROTECT	0 Code protect enabled 1 Code protect disabled	Specified code protection.	Disabled

For SX18/20/28AC Old Revision (before Rev 5.2), the fuse table is as follows:

Note: For backward compatibility, define OLDREV to indicate the SX18/20/28AC Part is earlier than Rev 5.2.

Fuse Settings for SX18/20/28AC Old Revision (Before Rev 5.2)	Descriptions	Function	Default
PINS18,OLDREV PINS20,OLDREV PINS28,OLDREV PINS48 PINS52	SX18AC Old Revision SX20AC Old Revision SX28AC Old Revision SX48BD SX52BD	Specifies device type and revision.	PINS18
PAGES1 PAGES2 PAGES4 PAGES8	512 words 1024 words 2048 words 2048 words	Specifies the number of program memory pages. This defines the program memory size.	PAGES8
BANKS1 BANKS2 BANKS4 BANKS8	1 bank 2 banks 4 banks 8 banks	Specifies the number of RAM banks.	BANKS8
OSCLP OSCXT OSCHS OSCRC	Ext Osc - LP Ext Osc - XT Ext Osc - HS Ext OSC - RC	Specifies external crystal / resonator Or external RC circuit.	OSCRC
IRCDIV1 IRCDIV2 IRCDIV4 IRCDIV8 IRCDIV16 IRCDIV32 IRCDIV64 IRCDIV128	Int Osc - 4MHz Int Osc - 2MHz Int Osc - 1MHz Int Osc - 500kHz Int Osc - 250kHz Int Osc - 125kHz Int Osc - 62.5kHz Int Osc - 31.25kHz	Specifies interal oscillator divider	4MHz
BROWNOUT	BOR at 4.2 volts	Specifies brown-out trigger at 4.2 volts	No brownout
TURBO	0 Turbo mode(1:1) 1 Compatible mode(1:4)	Specifies turbo mode	Compatible mode
STACKX	0 Stack is 8-levels 1 Stack is 2-levels	Specifies stack extension	2-level
OPTIONX	0 8-bit option register 1 6-bit option register	Specifies Option register extension	6 bits
CARRYX	1 ignore carry flag as input to ADD and SUB instruction	ADD and SUB instructions use Carry flag as input	Carry flag ignored
SYNC	0 Enable synchronous inputs 1 Disable synchronous inputs	Enable or Disable synchronous input mode(for turbo mode operation).	Disabled
WATCHDOG	0 Disable watchdog timer 1 Enable watchdog timer	Enable or Disable Watchdog Timer.	Disabled
PROTECT	0 Code protect enabled 1 Code protect disabled	Specified code protection.	Disabled

For SX48/52BD, the fuse table is as follows:

Fuse Settings for SX48/52BD)	Descriptions	Function	Default
PINS18/SX18AC PINS20/SX20AC PINS28/SX28AC PINS48/SX48BD PINS52/SX52BD	SX18AC SX20AC SX28AC SX48BD SX52BD	Specifies device type.	PINS18
OSCLP OSCXT OSCHS OSCRC	Ext Osc - LP Ext Osc - XT Ext Osc - HS Ext OSC - RC	Specifies external crystal / resonator Or external RC circuit.	OSCRC
IRCDIV1 IRCDIV2 IRCDIV4 IRCDIV8 IRCDIV16 IRCDIV32 IRCDIV64 IRCDIV128	Int Osc - 4MHz Int Osc - 2MHz Int Osc - 1MHz Int Osc - 500kHz Int Osc - 250kHz Int Osc - 125kHz Int Osc - 62.5kHz Int Osc - 31.25kHz	Specifies interal oscillator divider	4MHz
BROWNOUT BOROFF	Bits should not be changed unless brown-out refeature is to be disabled 11b - Disable BOR	Specifies brown-out reset	No brownout
TURBO	0 Turbo mode(1:1) 1 Compatible mode(1:4)	Specifies turbo mode	Compatible mode
STACKX	0 Stack is 8-levels 1 Stack is 2-levels	Specifies stack extension	2-level
OPTIONX	0 8-bit option register 1 6-bit option register	Specifies Option register extension	6 bits
CARRYX	1 ignore carry flag as input to ADD and SUB instruction	ADD and SUB instructions use Carry flag as input	Carry flag ignored
SYNC	0 Enable synchronous inputs 1 Disable synchronous inputs	Enable or Disable synchronous input mode(for turbo mode operation).	Disabled
WATCHDOG	0 Disable watchdog timer 1 Enable watchdog timer	Enable or Disable Watchdog Timer.	Disabled
PROTECT	0 Code protect enabled 1 Code protect disabled	Specified code protection.	Disabled
BROWNOUT BOROFF	Bits should not be changed unless brown-out refeature is to be disabled 11b - Disable BOR	Specifies brown-out reset	No brownout
SLEEPCLK	0 Enable clock operation during sleep mode 1 Disable clock operation during sleep mode	Sleep Clock Disable	Disable sleep clock

WDRT60 WDRT480 WDRT960 WDRT1920 WDRT0 WDRT006 WDRT768 WDRT184	60 msec 480 msec 960 msec 1920 msec 0msec(no delay) 0.06 msec 7.68 msec 18.4 msec(default)	Delay Reset Timer timeout period	18.4msec

ID - Set an ID string in program memory

```
Syntax:          ID "Text"
Description:     Assigns an ID text string at the end of program memory.
                 The string may be up to 8 characters and should be in quotes
Example:         ID 'Demo28"
```

RESET - Set reset vector address

```
Syntax:          RESET <expression> [<comment>]
Description:     Put the instruction opcode [JMP Start] at the reset vector
                 memory location. The reset vector values depend on configured
                 memory size on chip. The reset vector is default at 0x7FF.

                 Define PAGESx in FUSES  Reset vector
                 ======================================
                 FUSES PAGES1            0x1FF
                 FUSES PAGES2            0x3FF
                 FUSES PAGES4            0x7FF
                 FUSES PAGES8            0x7FF

Example:         DEVICE      PINS18
                 RESET       Start

                 This is equivalent to:
                 ORG   1FFh
                 JMP   Start
```

EQU - Equate a symbol to an expression

```
Syntax:          <label> EQU <expression> [<comment>]
Description:     A constant value or the value of a well-defined expression is
                 assigned to the given label. Note that any value defined with
                 an EQU directive is fixed and may not be redefined.
Example:         COUNT EQU   19h
```

SET or = - Set a symbol equal to an expression

```
Syntax:          [<label>] SET <expression> [<comment>]
Description:     To assign the value of a well-defined expression to a label.
                 Unlike the EQU directive, SET can be used more than once on
                 the same symbol; with the most recent SET statement
                 determining the value of the label.
Example:         FIVE  SET   5
                 or
                 FIVE  =     5
```

DS - Define memory space

```
Syntax:         [<label>] DS <operand>

Description:    Define memory space by incrementing the program memory address
                during assembly.
Example:        ORG             $10
                Timers          =     $
                timers_low      ds    1     ; $10
                timers_high     ds    1     ; $11
                timer_accl      ds    1     ; $12
                timer_array     ds    3     ; $13, $14, $15
```

DW - Define data in memory

```
Syntax:         [<label>] DW <operand>

Description:    Initialize one or more words of program memory with data. The
                data may be in the form of constants or ASCII character
                strings.
Example:        DW   10h,20h,30h
                Or
                DW   'This is a test'
```

RES - Reserve storage in memory

```
Syntax:         [<label>] RES  <expression> [<comment>]
Description:    The program counter will be advanced by the amount of
                the expression.
Example:        RES    10
```

INCLUDE - Insert external source file

```
Syntax:         [<label>] INCLUDE "<filename>" [<comment>]
Description:    To read in the specified file as source code. A path name
                can be provided if the file resides in another directory.
Example:        INCLUDE  "SXREG.INC"
```

ORG - Set program origin

```
Syntax:         [<label>] ORG <expression> [<comment>]
Description:    Set program origin for subsequent code at the address defined
                in constant value.
Example:        ORG   0
                Or
                ORG   $100
```

IF.ELSE.ENDIF - Conditional assembly

Syntax:
```
IF <expression>
      <source lines>
ELSE
      <source lines>
ENDIF
```
Description: This directive is used to creat conditional assembly blocks. ELSE is used in conjunction with IF directive to provide an alternative path. If IF tests false, the alternative path noted by the ELSE directive is taken, providing conditional assembly. The IF statement requires a matching ENDIF statement.

Example:
```
count equ   12h
IF (count > 10h)
      INC   4
ELSE
      DEC   4
ENDIF
```

IFDEF.ELSE.ENDIF - Conditional assembly

Syntax:
```
IFDEF <symbol>
      <source lines>
ELSE
      <source lines>
ENDIF
```
Description: ELSE is used in conjunction with IFDEF directive to provide an alternative path. If symbol is not defined, the alternative path noted by the ELSE directive is taken, providing conditional assembly. The IFDEF statement requires a matching ENDIF statement.

Example:
```
var1  equ   10h
.
.
.
IFDEF  var1
      INC   4
ELSE
      DEC   4
ENDIF
```

IFNDEF.ELSE.ENDIF - Conditional assembly

Syntax:
```
IFNDEF <symbol>
      <source lines>
ELSE
      <source lines>
ENDIF
```
Description: ELSE is used in conjunction with IFNDEF directive to provide an alternative path. If symbol is defined, the alternative path noted by the ELSE directive is taken, providing

```
                conditional assembly. The IFNDEF statement requires a matching
                ENDIF statement.
Example:        IFNDEF  var1
                      INC   4
                ELSE
                      DEC   4
                ENDIF
```

REPT-ENDR – Repeat Code Block

```
Syntax:         REPT  count
                Codeblock
                ENDR
Description:    Use to indicate a block of code to be repeated a
                Specified number of times during assembly.
Example:        REPT  3
                add $12,#1
                ENDR

      will be expanded to
                add   $12,#1
                add   $12,#1
                add   $12,#1
                during program assembly.

      Within the block, the % sign may be used to refer to the
      current iteration(1-n), i.e. % equal to 1 the first time through
      the repeat block, % equal to 2 the second time through the loop,etc.
      For example:
                REPT  3
                Add   $12,#%
                ENDR

      will be expaned to
                Add   $12,#1
                Add   $12,#2
                Add   $12,#3
      during assembly.
```

LPAGE - Insert page eject in listing file

```
Syntax:         [<label>] LPAGE [<comment>]
Description:    Insert a form feed at this point in the listing file.
Example:        LPAGE
```

SPAC - Insert lines in listing file

```
Syntax:         [<label>] SPAC <expression> [<comment>]
Description:    Insert the number of blank lines given by the expression
                into the listing file.
Example:        SPAC  5
```

TITLE - Define program heading

```
Syntax:          [<label>]  TITLE "<string>"  [<comment>]
Description:     Set up the text to be used in top line of listing file.
Example:         TITLE    "SAMPLE.ASM"
```

END - End of source program

```
Syntax:          [<label>]   END  [<comment>]
Description:     Mark the end of source program.
Example:         END        ; terminate the program
```

Note: If you have declared the same directive more than once, the latter one will overwrite the previous definition.

CHAPTER 4: Using Macros

Macros consist of sequences of assembler instructions and directives that can be inserted in the assembly source code by using a macro call. The macro must first be defined, then it can be call upon later at any point within the source program.

Macro Definition

A macro definition is divided into three areas:

macro heading,
macro body, and
macro terminator.

Macro Heading

The format of the macro heading is as follows:

```
<label>  MACRO  [<parameter> ... <parameter>]   [<comment>], where
```

Label is the name of the macro, and
Parameter is an input argument passed into the macro call by value. Parameter can only be operand values, not instructions.

OR

```
<label>  MACRO  {Argcount}
         codeblock
         {EXITM}
         ENDM where
```

Argcount specifies the exact number of argments required by the macro and must range from 1 to 64.

NOTE: The comment field is permitted in the heading whether or not there are parameters.

The name of the macro must comply with SASM label rules. If a macro name is identical to a mnemonic or an assembler directive, the assembler will generate an error.

Macro Body

The macro body begins immediately after the macro definition and continues until the macro terminator. The macro body consists of a sequence of source lines that may contain a formal parameter in any field. When the macro is instantiated, all parameters will be replaced by the corresponding arguments provided by the macro call.

Macro Terminator

The ENDM directive terminates the macro definition. ENDM must exist before another MACRO statement is found. The format of the macro terminator is as follows:

```
ENDM [<comment>]
```

Macro Call

Once the macro has been defined, it can be instantiated at any point within the source program by using a macro call as described below:

[<label>] <name> [<arg> [,<arg>] ...] [<comment>] where

<label> is assigned the current value of the location counter
<name> is the name of the macro to be instantiated<arg> is any symbol or constant passed as a parameter to the macro

The macro call itself will not occupy any locations in memory. However, the macro instantiation will begin at the current memory location. The argument list is terminated by white space or a semi-colon.

Parameters

All arguments are passed into the macro instantiation by value. Currently SASM supports symbols, constants and fr.bit type as macro parameters. However, it does not allow string operands or reserved symbols as macro arguments.

Local Symbols

Local symbols are labels declared within macros only. These symbols are local to the particular macro and are differentiated from regular labels which are global to the entire program. Each time the macro is called, SASM will assign each local symbol a system generated symbol of the form ??0001, ??0002, ??0003. etc. All Local definitions must occur immediately after the MACRO heading and before the first line of the macro body with a syntax as followed:

These local macro labels do not have to start at column 1, as the global labels.

LOCAL <label> [,<label>] ...

Macro Example1

Definition

```
; Define macro
CLRREG      MACRO reg
            LOCAL again
again       clr   reg
            jmp   again
            ENDM
```

Macro Call

```
CLRREG      20h
```

Macro expansion

```
0046                                CLRREG      08h
0046  0006  0068  m ??0000         clr   08h
0046  0007  0A06  m                jmp   ??0000
0047
```

Macro Example2

Definition

```
; Define macro
ANDREG      MACRO 3
            And   W,#1
            And   W,#2
            And   W,#3
            ENDM
```

Macro Call

```
ANDREG      5,6,7
```

Macro expansion

```
0046                                ANDREG      5,6,7
0046  0006  0E05  m                and   W,#5
0046  0007  0E06  m                and   W,#6
0047  0008  0E07  m                and   W,#7
```

CHAPTER 5: SASM Assembler Output Files

When SASM is activated, you will see the following:

```
SASM Cross-Assembler for Scenix SX-based Microcontrollors        Version 1.35
Copyright(c) Advanced Transdata Corporation 1998-1999

xxx lines compiled in xxx seconds
xxx symbols
< error status >
```

For each source file submitted, the SASM will produce the following files:
HEX: object file
LST: listing file, unless the /L switch is given to suppress its output
SYM: symbol file
MAP: map file
ERR: error message file

Object File (HEX or OBJ)

The object file can be in different formats and contains data that can be loaded and executed. SASM outputs INHX8M (Intel 8-bit Hex file) format as the default. This file will be used by PGM-SX Programmer and the SX-ISD Debugger for programming the SX devices.

The other formats: BIN16, INHX16, and INHX8S are provided to support other programmers. Please refer to Appendix A for more information on the individual object file formats.

Listing File (LST)

The listing file contains the source code along with some useful information about the output addresses and corresponding object code. Each line from the source code will be reproduced in the listing file and accompanied by the listing file line number, program counter and the object code (OPCODE).

Example

```
LINE  PC    OPCODE
0011  0000  0C02   mov     W,#00000010b
0012  0001  01A6   xor     rb,W            ;toggle rb.1
0013  0002  0CEC   mov     w,#-20
0014  0003  000F   retiw
```

The first field is a 4-digit decimal number that represents the line number at which the source line appears in the source code. The second field is a 4-digit hex number that represents the current program counter. The third field is a 4-digit hex number that represents the opcode generated from the source line. This is the actual value that will appear in the object code.

Cross Reference Listing

A cross-reference table is generated at the end of the listing file. This table contains a list of every symbol used in the source file along with its symbol type, value and the source line number.

For example:

```
SYMBOL      TYPE  VALUE LINE
W           RESV  0000  0006
LOOPB       ADDR  0109  0044
XCNT        DATA  0010  0011
YCNT        VAR   0011  0045
```

Where

DATA	user-defined symbol that represents a data variable defined by EQU directive
VAR	user-defined symbol that represents a data variable defined by SET directive
ADDR	user-defined symbol that represents a code address or a program counter location
RESV	predefined symbol used internally by SASM

Symbol File (SYM)

The symbol file is identical to the cross reference portion of the listing file. It lists all symbols found in the source file, provides information on their type, value and the specific line numbers where they are found. The symbol file, generated with the /D switch, is required to define watch variables and to specify breakpoint at address label for Transdata's emulators.

Map File (MAP)

The map file contains line correspondence between source file, program counter and file number. This file is necessary to enable source level debugging with Transdata's emulators. The contents of the map file vary, depending on which switch is used during compilation.

SASM generates correspondence between source file (.ASM) and program counter. It enables Emulators to load the source file to the Source Window during debugging.

Error File (ERR)

The error file contains all error messages generated during program compilation. If there is no error, the file will have zero byte.

Error Messages

Error messages are displayed at the terminal and in the listing file. They all have the following format:

<List Line#><File(Source Line#)><Error/Warning Count>:<Pass#>:<message>

A list of error/warning messages is given in Appendix C.

APPENDIX A: Summary of SX-based Instruction Set

Mnemonics,Operands	Flags	Description	Microchip Equivalent
Logical Operations			
AND fr,W	Z	AND W into fr	ANDWF fr,1
AND W,fr	Z	AND fr into W	ANDWF fr,0
AND W,#lit	Z	AND literal into W	ANDLW lit
NOT fr	Z	One's complement of fr into fr	COMF fr,1
NOT W	W,Z	One's complement of W into W	XORLW 0FFh
OR fr,W	Z	OR W into fr	IORWF fr,1
OR W,fr	Z	OR fr into W	IORWF fr,0
OR W,#lit	Z	OR literal into W	IORLW lit
XOR fr,W	Z	XOR W into fr	XORWF fr,1
XOR W,fr	Z	XOR fr into W	XORWF fr,0
XOR W,#lit	Z	XOR literal into W	XORLW lit
Arithmetic and Shift Operations			
ADD fr,W	C,DC,Z	Add W to fr into fr	ADDWF fr,1
ADD W,fr	C,DC,Z	Add fr to W into W	ADDWF fr,0
CLR fr	Z	Clear fr to 0	CLRF fr
CLR W	Z	Clear W to 0	CLRW
CLR !WDT	TO,PD	Clear WDT and prescaler	CLRWDT
DEC fr	Z	Decrement fr	DECF fr,1
DECSZ fr	-	Decrement fr, skip if zero	DECFSZ fr,1
INC fr	Z	Increment fr	INCF fr,1
INCSZ fr	-	Increment fr, skip if zero	INCFSZ fr,1
NOP	-	No operation	NOP
RL fr	C	Rotate left fr into fr	RLF fr,1
RR fr	C	Rotate right fr into fr	RRF fr,1
SUB fr,W	C,DC,Z	Subtract W from fr	SUBWF fr,1
SWAP fr	-	Swap nibbles in fr into fr	SWAPF fr,1
Bitwise Operations			
CLRB bit	-	Clear bit to 0	BCF bit
CLC	C	Clear carry	BCF 3,0
CLZ	Z	Clear zero	BCF 3,2
SB bit	-	Skip if bit = 1	BTFSS bit
SETB bit	-	Set bit to 1	BSF bit
SNB bit	-	Skip if bit = 0	BTFSC bit
Data Movement Operations			
MOV fr,W	-	Move W into fr	MOVWF fr
MOV W,fr	Z	Move fr into W	MOVF fr,0
MOV W,fr-W	C,DC,Z	Move fr-W into W	SUBWF fr,0
MOV W,#lit	-	Move literal into W	MOVLW lit
MOV W,/fr	Z	Move 1's complement of fr to W	COMF fr,0
MOV W,--fr	Z	Move fr-1 into W	DECF fr,0
MOV W,++fr	Z	Move fr+1 into W	INCF fr,0
MOV W,<<fr	C	Move left-rotated fr into W	RLF fr,0
MOV W,>>fr	C	Move right-rotated fr into W	RRF fr,0
MOV W,<>fr	-	Move nibble-swapped fr into W	SWAPF fr,0
MOV W,M	-	Move MODE into W	-
MOV M,W	-	Move W into MODE	-
MOV M,#lit	-	Move literal into MODE	

```
MOV    !rx,W          -      Move W into Port Rx control register    TRIS
MOV    !OPTION,W      -      Move W into OPTION                      OPTION
MOVSZ  W,--fr         -      Move fr-1 into W, skip if zero          DECFSZ fr,0
MOVSZ  W,++fr         -      Move fr+1 into W, skip if zero          INCFSZ fr,0
SC                    C      Skip if carry bit is set                BTFSS 3,0
TEST   fr             Z      Test if fr equal to 0                   MOVF fr,1
```

Control Transfer Operations

```
CALL   addr8          -      Call to address                         CALL addr
JMP    addr9          -      Jump to address                         GOTO addr
JMP    W              -      Move W into PC(L)                       MOVWF 2
JMP    PC+W           C,DC,Z Add W into PC(L)                        ADDWF 2,1
RET                   -      Return from call without                -
                             affecting W
RETP                  -      Return from call,                       -
                             write to PA2:PA0
RETI                  -      Return from interrupt                   -
RETIW                 -      Return from interrupt,                  -
                             subtract W from RTCC
RETW   #lit           -      Return from call, move                  RETLW  lit
                             literal in W
SKIP                  -      Skip the following instruction          BTFSC/BTFSS 2,0
```

System Control Operations

```
BANK   n              -      Transfer n to FSR7:FSR5                 -
IREAD                 -      Read instruction at MODE:W              -
                             into MODE:W
MODE   n              -      Transfer n into MODE                    -
M      n              -      Transfer n into MODE                    -
PAGE   n              -      Transfer n to PA2:PA0                   -
SLEEP                 TO,PD  Clear WDT and enter sleep mode          SLEEP
```

APPENDIX B: Object File Format

Intel Hex file formats

This is the most commonly used format for file interchange with EPROM programmers. A complete Intel Hex file contains one or more hexadecimal records. The file ends with an end of file record.

Each data record begins with a nine-character prefix and ends with a two-character checksum. Each letter corresponds to one hexadecimal digit in ASCII representation.

Example :BBAAAATTHHHH...HHHCC

Definitions

:	Record start character
BB	Byte count - the hexadecimal number of data bytes in the record.
AAAA	Load address in hexadecimal of first data byte in this record.
TT	Record type. The record type is 00 for data records and 01 for the end record.
HH	One hexadecimal data byte.
CC	Record checksum. This is the 2's complement of the summation of all the bytes in the record from the byte count through the last byte. While the summation is calculated, it is always truncated to a one byte result.

INHX8M: Merged 8-bit Intellex Hex Format

This is the default hex file that will be generated by the SASM cross assembler.

This format produces one 8-bit Hex file with a low-byte/high-byte combination. Since each address can only contain 8 bits in this format, all addresses will be doubled. File extensions for the object code will be '.HEX'.

Example:

```
:0800000001024307000 8640C33
:080008002100A502040000081C
:08020000000C0500250026009A
:080208000600030C0200640C67
:080210002100A6020A0C3000D7
:080218000009F0020C0B090BB8
:08FFE000000000000000000019
:043FFE00FF0FFF0FA3
:00000001FF
```

INHX16: 16-bit Hex Format

This format will be output if the INHX16 option is used with the LIST F directive or with the '/f' option on the command line. This format produces one 16-bit Hex file with a high-byte/low-byte combination. File extension for the object code will be '.HEX'.

Example:

```
:0800000002010743080000C64002102A5000408005F
:080100000C0000050025002600060C0300020C6414
:08010800002102A60C0A0030090002F00B0C0B09BA
:047FF000000000000000000008D
:021FFF000FFF0FFFC4
:00000001FF
```

INHX8S: Split 8-bit Intellec Hex file format

This format will be output if the INHX8S option is used with the LIST F directive or with the '/f' option on the command line.

This format produces two 8-bit Hex files, one containing the address/data pairs for the high order 8 bits and the other will contain the low-order 8 bits. File extensions for the object code will be '.HXL' and 'HXH' for low and high order files respectively.

Example:

SAMPLE.HXL:

```
:08000000143006421A5040086
:0801000000052526060302643 8
:0801080021A60A3000F00C09E9
:047FF000000000008D
:021FFF00FFFFE2
:00000001FF
```

SAMPLE.HXH:

```
:080000000207080C00020008D1
:080100000C000000000C000CD3
:0801080000020C0009020B0BC0
:047FF000000000008D
:021FFF000F0FC2
:00000001FF
```

Binary file Format

This format will be output if the BIN16 option is used with the LIST F directive or with the '/f' option on the command line. A pure 16-bit binary file will be generated. A screen dump of SAMPLE.OBJ using DEBUG will be as follows:

```
Debug SAMPLE.OBJ
-d 300 32f

1846:0100  01 02 43 07 00 08 64 0C-21 00 A5 02 04 00 00 08   ..C...d.!.......
1846:0110  FF 0F FF 0F FF 0F FF 0F-FF 0F FF 0F FF 0F FF 0F   ................
1846:0120  FF 0F FF 0F FF 0F FF 0F-FF 0F FF 0F FF 0F FF 0F   ................
1846:0130  FF 0F FF 0F FF 0F FF 0F-FF 0F FF 0F FF 0F FF 0F   ................
1846:0140  FF 0F FF 0F FF 0F FF 0F-FF 0F FF 0F FF 0F FF 0F   ................
```

APPENDIX C: SXREG.INC Definition File

The SXREG.H file contains the definition of the file registers for the SX microcontrollers. This file can be conveniently incorporated into the source file using the INCLUDE statement.

```
      INCLUDE     "SXREG.INC"
```

```
;******************** SX Registers ********************
;
; Registers
IND     equ     0
INDF    equ     0
RTCC    equ     1
WREG    equ     1
PC      equ     2
PCL     equ     2
STATUS  equ     3
FSR     equ     4
RA      equ     5
RB      equ     6
RC      equ     7
RD      equ     8
RE      equ     9

; Status File Register Bits
C       equ     STATUS.0
DC      equ     STATUS.1
Z       equ     STATUS.2
PD      equ     STATUS.3
TO      equ     STATUS.4
PA0     equ     STATUS.5
PA1     equ     STATUS.6
PA2     equ     STATUS.7
;
;*****************************************************************
```

APPENDIX D: Error Message

```
1     Bad instruction statement
2     Redefinition of symbol
3     Symbol is not defined
4     Symbol is a reserved word
5     Missing operand(s)
6     Too many operands
7     Missing file register
8     Missing literal
9     Missing Label
10    Missing right parenthesis
11    Missing expression
12    Redefinition of MACRO label
13    Bad expression
14    Bad argument
15    Bad MACRO expression
16    Macro argument do not match
17    Unmatched MACRO
18    Bad IF-ELSE-ENDIF statement
19    Unmatched ELSE
20    Unmatched ENDIF
21    File nesting error - too deep
22    If.else.endif nesting error - too deep
23    Bad numeric string format
24    Value is out of range
25    Bad radix value
26    Unknown microcontroller type
27    Unknown output format
28    Unknown listing parameter
29    Bad string syntax
30    Overwriting same program counter location
31    Expected an \'=\' sign
32    Unexpected EOF
33    Assume value is in HEXADECIMAL
34    Token length exceeds limit
35    Illegal character - Ignored
36    File register truncated to 5 bits
37    Literal truncated to 8 bits
38    Missing RAM Bank bits
39    No destination bit
40    Destination bit can only be 0 or 1
41    Bit number out of range
42    Address change across page boundary
43    Address exceeds memory limit
44    Address is not within lower half of memory page
45    Label must begin at column 1
```